


Dokumentation

zu
Programm zum Datei-gesteuerten Schreiben von Magnetkennkarten

Bearbeiter: Pr 

EKG/Abt. DC

Version : 1.0

1. Kurzbeschreibung

Das Programm dient zum Datei-gesteuerten Schreiben von Magnetkennkarten mit der Schreib-Lese-Einrichtung (SLE) an der Kopfstation K 8915 K. Die Daten fuer die Kennkarten koennen aus einer Textdatei oder einer dBase-2-Datei entnommen werden.
Es wird nur die 2. Spur der Kennkarte bearbeitet.

2. Aufruf des Programmes

Syntax:

```
KKARTE  --/m-----  
          1) Menue-gesteuertes Lesen und Schreiben  
          |--/t-Ueberschrift---TextName-----  
          2) Schreiben aus der Textdatei  
          |--/d-Ueberschrift--dBaseName--FeldAnzeige--FeldKennkarte-----  
                                                  |--FeldBedingung--|  
          3) Schreiben aus dBase-Datei
```

2.1 Menue-gesteuertes Lesen und Schreiben

Parameter:

Flag: /m

Es wird vor jeder Aktion Lesen oder Schreiben ausgewaehlt. Beim Lesen wird der Karteninhalt angezeigt. Vor dem Schreiben muss der Karteninhalt eingegeben werden.

2.2 Schreiben aus einer Textdatei

Parameter:

Flag: /t

Ueberschrift : ist Ueberschrift fuer die Sitzung; Leerzeichen durch "-" darstellen.

TextName : Name der Textdatei

Aufbau der Textdatei:

- je Zeile wird eine Kennkarte geschrieben
- jede Zeile muss aus mind. 2 Teilen bestehen
- die Teile muessen mit "," getrennt sein; Strings koennen mit "\"" begrenzt sein (dBase-Format: SDF Delimiter)
- der erste Teil ist die Beschriftung der Karte (max 80 Zeichen)
- der zweite Teil ist der Kennkarten-Inhalt

2.3 Schreiben aus einer dBase-2-Datei

Parameter:

Flag: /d

Ueberschrift : ist Ueberschrift fuer die Sitzung; Leerzeichen durch "-" darstellen.

FeldBeschriftung: Feldname der Beschriftung der Kennkarte

FeldKennkarte : Feldname des Karteninhaltes

Feldbedingung : wenn angegeben dann

wenn dieses Feld an erster Position kein Leerzeichen ist,
dann wird der Datensatz benutzt und das Feld geloescht
sonst wird der Datensatz uebergangen.
sonst werden alle Datensaeetze benutzt.

3. Funktion des Programmes

```

program KKarte
begin
  {Parameter einlesen}
  {Bildschirm aufbauen}
  {Datensätze zählen (nur bei 2 und 3)}
  if {kein Dateifehler und Daten vorhanden} then
    repeat
      {Daten holen      : Funktion abhaengig vom Flag}
      {Karte bearbeiten: Funktion abhaengig vom Flag}
    until {alle Daten bearbeitet}
    {Datei schleissen}
  endif
end.

```

4. Quelltexte geschrieben in Turbo-Pascal 3.0 (CP/M 80)

- | | |
|-----------------|---|
| (1) SL.INL | Treiber fuer die SLE als MC-Programm in der Form von Inline-Anweisungen |
| (2) KKDef.INC | Definitionen der Typen, Konstanten und globale Variablen |
| (3) KKSle.INC | Prozeduren zum Aufruf des Treibers |
| (4) KKBild.INC | Bildschirm Aufbau, Fehlermeldungen und Bedienerfuehrung |
| (5) dBase2.INC | Zugriff zu einer dBase-2-Datei |
| (6) TxtDat.INC | Zugriff zu den Teilen der Textdatei |
| (7) KKMenue.INC | menuegesteuerte Auswahl und Dateneingabe |
| (8) KKPar.INC | Lesen und Pruefen der Parameter |
| (9) KKArb.INC | Bearbeitungsprogramm |
| (10) KKarte.PAS | Hauptprogramm |

4.1 SL.INL

Die Inline-Datei wird aus einer ".PRN"-Datei mit dem Programm PrnInl.EXE erzeugt (Turbo-Pascal 4.0 DCP). Sie wird als erste Prozedur des Programmes eingebunden (in KKarte.PAS):

```

procedure SLE;           {Treiber am Anfang}
begin
  inline (
    {#i SL.inl}           {MC-Programm ab 20FFh !}
  );
end{SLE};

```

Das INL-Programm muss deshalb auf die Adresse 20FFh gebunden sein (entweder schon im .PRN oder binden mit PrnInl). Die Puffer zum Lesen und Schreiben sowie fuer die Fehlermeldungen werden als absolute Variable und die Prozeduren zum Lesen und Schreiben als external in KKSle.INC definiert.

4.2 dBase2.INC

Diese Include-Quelle ist allgemein aufgebaut und kann fuer andere Zwecke verwendet werden. Es koennen dBase-2-Dateien feldweise gelesen und geschrieben, aber keine neuen Datensätze angehaengt werden. Wegen der Puffergroesse ist die Datensatzlaenge auf maximal 2040 Byte beschraenkt. Es werden folgende Prozeduren realisiert:

- dBOpen : Eroeffnen einer dBase-2-Datei mit Test des Dateikopfes und Einlesen des ersten Datensatzes.
- dBRead : Lesen eines vorgegebenen Feldinhaltes des aktuellen Satzes
- dBWrite : Schreiben in ein vorgegebenen Feldinhaltes des aktuellen Satzes
- dBSkip : Positionieren auf den naechsten Datensatzes mit Uebergehen geloeschter Saetze.
- dBEOF : Test, ob Dateiende erreicht.
- dBClose : Schliessen der Datei.

4.3 TxtDat.INC

Diese Include-Quelle ist allgemein aufgebaut und kann fuer andere Zwecke verwendet werden. Es koennen Datendateien im Textformat gelesen werden, die folgenden Aufbau besitzen:

- jede Zeile kann aus mehreren Teilen bestehen
- die Teile muessen mit "," getrennt sein; Strings koennen mit "'" begrenzt sein (dBase-Format: SDF Delimiter)

Es werden folgende Prozeduren realisiert:

- TxtOpen : eroeffnen der Textdatei und Einlesen der ersten Zeile.
- TxtRead : Lesen eines vorgegebenen Datensatzteiles.
- TxtSkip : Einlesen der naechsten Zeile.
- TxtEof : Test auf Dateiende.


```

(22C0)/$7A
(22C1)/$B3
(22C2)/$20/$FB
(22C4)/$C9
(22C5)/$F5
(22C6)/$11/>>$0280
(22C9)/$1B
(22CA)/$7A
(22CB)/$B3
(22CC)/$20/$FB
(22CE)/$3E/$43
(22D0)/$D3/$96
(22D2)/$F1
(22D3)/$C9

```

```

( LD A,D
( OR E
( JR NZ,WAITS1
( RET
(WAIT: PUSH AF
(WAIT1: LD DE,$0280H
( DEC DE
( LD A,D
( OR E
( JR NZ,WAIT1
( LD A,$3H
( OUT (PIOMOD),A
( POP AF
( RET

```

Warteschleife 2
6,8ms

Schreibroutine

```

(22D4)/$CD/>>$24E9
(22D7)/$CD/>>$250D
(22DA)/$3A/>>$21A8
(22DD)/$FE/$53
(22DF)/$CA/>>$2402
(22E2)/$0E/$09
(22E4)/$11/>>$2601
(22E7)/$CD/>>$0005
(22EA)/$21/>>$210D
(22ED)/$06/$25
(22EF)/$E5
(22F0)/$C5
(22F1)/$0E/$01
(22F3)/$CD/>>$0005
(22F6)/$CD/>>$243F
(22F9)/$28/$08
(22FB)/$3E/$03
(22FD)/$32/>>$2109
(2300)/$D3/>>$2448
(2303)/$C1
(2304)/$E1
(2305)/$77
(2306)/$FE/$0D
(2308)/$28/$03
(230A)/$23
(230B)/$10/$E2
(230D)/$21/>>$210B
(2310)/$3E/$0D
(2312)/$01/>>$0026
(2315)/$ED/$B1
(2317)/$28/$02
(2319)/$18/$06
(231B)/$2B
(231C)/$41
(231D)/$77
(231E)/$23
(231F)/$10/$FC
(2321)/$3E/$1F
(2323)/$77
(2324)/$21/>>$210B
(2327)/$01/>>$0025
(232A)/$3E/$20
(232C)/$ED/$B1
(232E)/$20/$06
(2330)/$2B
(2331)/$3E/$0D
(2333)/$77
(2334)/$18/$EE
(2336)/$06/$25
(2338)/$21/>>$210B
(233B)/$7E
(233C)/$E6/$0F
(233E)/$E2/>>$2343
(2341)/$C6/$10
(2343)/$77
(2344)/$23
(2345)/$10/$F4
(2347)/$21/>>$210B
(234A)/$AF
(234B)/$F5
(234C)/$0E/$0B
(234E)/$F1
(234F)/$A9
(2350)/$F5
(2351)/$7E
(2352)/$E6/$0F
(2354)/$C6/$30
(2356)/$FE/$3D
(2358)/$28/$0B
(235A)/$FE/$3F
(235C)/$28/$0B
(235E)/$4F
(235F)/$F1
(2360)/$A9
(2361)/$F5

```

```

(SCHREIB: CALL INIB
( CALL INI9
( LD A,(MERK)
( CP 53H
( JP Z,SCHREIBB
( LD C,9
( LD DE,LER
( CALL BDOS
( LD HL,PUFE
( LD B,37
(SCHREIB1: PUSH HL
( PUSH BC
( LD C,1
( CALL BDOS
( CALL PRUEF
( JR Z,SCHR
( LD A,3
( LD (FESP),A
( JP FEHLER
(SCHR: POP BC
( POP HL
( LD (HL),A
( CP 00DH
( JR Z,SCHREIB2
( INC HL
( DJNZ SCHREIB1
(SCHREIB2: LD HL,PUFE
( LD A,00DH
( LD BC,38
( CPIR
( JR Z,SCHREIB3
( JR SCHREIBF
(SCHREIB3: DEC HL
( LD B,C
( LD (HL),A
( INC HL
( DJNZ SCHREIB4
(SCHREIBF: LD A,1FH
( LD (HL),A
( LD HL,PUFE
( LD BC,37
( LD A,20H
( CPIR
( JR NZ,SCHREIBX
( DEC HL
( LD A,00DH
( LD (HL),A
( JR SCHREIBU
(SCHREIBX: LD B,37
( LD HL,PUFE
(SCHREIBY: LD A,(HL)
( AND 0FH
( PD,SCHREIBW
( ADD A,10H
(SCHREIBW: LD (HL),A
( INC HL
( DJNZ SCHREIBY
( LD HL,PUFE
( XOR A
( PUSH AF
( LD C,00BH
( POP AF
( XOR C
( PUSH AF
( LD A,(HL)
( AND 0FH
( ADD A,30H
( CP 3DH
( JR Z,$SUM0
( JR Z,$SUM1
($SUM0: LD C,A
( POP AF
( XOR C
( PUSH AF

```

CTC-Init

bei nicht Test

Leerzeile ausgeben

Schreibspeicher
37 Zeichen festgelegt
Routine Schreibspeicher
fuellen
BDOS-Eingabe

unerlaubte Zeichen

CR ? (als Abschluss der Eingabe)?

CR suchen

mit Hex 0d auffuellen

Endekennzeichen

37 Zeichen fuer Karte
auf 5 Bits mit richtiger
Paarigkeit korrigieren

Pruefbyte errechnen

```

(2362)/$13/ INC HL
(2363)/$18/$EC JR QSUM
(2365)/$3E/$20 LD A,20H
(2367)/$18/$F5 JR QSUM00
(2369)/$E6/$0F (QSUM1: AND 0FH
LD C,A
(236B)/$4F POP AF
(236C)/$F1 XOR C
(236D)/$A9 AND 0FH
(236E)/$E6/$0F auf 5 Bits und richtige
(2370)/$E2/$2375 JR PO,QSUM2 Paaritaet korrigieren
(2373)/$C6/$10 ADD A,10H
(2375)/$32/$21AB (QSUM2: LD (PUFZ),A
(2378)/$DB/$94 IN A,(P100KD)
(237A)/$FE/$01 CP 1
(237C)/$28/$08 JR Z,SCHREIB5
(237E)/$3E/$02 LD A,2
(2380)/$32/$2109 LD (FESP),A
(2383)/$D2/$2448 JP NC,FEHLER
(2386)/$C0/$24F2 (SCHREIB5: CALL INI1
(2389)/$C0/$2526 CALL INI22
(238C)/$3E/$42 LD A,42H
(238E)/$D3/$96 OUT (P100D),A
(2390)/$3E/$00 LD A,80H
(2392)/$D3/$93 OUT (S102C),A
(2394)/$DB/$94 (SCH1: IN A,(P100KD)
(2396)/$FE/$06 CP 6
(2398)/$20/$FA JR NZ,SCH1
(239A)/$3E/$00 LD A,0BH
(239C)/$2F CPL
(239D)/$E6/$1F AND 1FH
(239F)/$D3/$92 OUT (S102D),A
(23A1)/$3E/$C5 LD A,9C5H
(23A3)/$D3/$93 OUT (S102C),A
(23A5)/$3E/$00 LD A,000H
(23A7)/$D3/$93 OUT (S102C),A
(23A9)/$21/$210B LD HL,PUFE
(23AC)/$06/$26 LD B,38
(23AE)/$3E/$42 (SCHREIB7: LD A,42H
(23B0)/$D3/$96 OUT (P100D),A
(23B2)/$7E LD A,(HL)
(23B3)/$F5 PUSH AF
(23B4)/$2F CPL
(23B5)/$E6/$1F AND 1FH
(23B7)/$D3/$92 OUT (S102D),A
(23B9)/$C0/$22BC CALL WAITS
(23BC)/$23 INC HL
(23BD)/$F1 POP AF
(23BE)/$FE/$1F CP 1FH
(23C0)/$28/$05 JR Z,SCHREIB8
(23C2)/$C0/$22C5 CALL WAIT
(23C5)/$10/$E7 DJNZ SCHREIB7
(23C7)/$C0/$22C5 (SCHREIB8: CALL WAIT
(23CA)/$21/$21AB LD HL,PUFZ
(23CD)/$06/$05 LD B,5
(23CF)/$3E/$42 (SCHREIB9: LD A,42H
(23D1)/$D3/$96 OUT (P100D),A
(23D3)/$7E LD A,(HL)
(23D4)/$2F CPL
(23D5)/$E6/$1F AND 1FH
(23D7)/$D3/$92 OUT (S102D),A
(23D9)/$23 INC HL
(23DA)/$C0/$22BC CALL WAITS
(23DD)/$C0/$22C5 CALL WAIT
(23E0)/$10/$ED DJNZ SCHREIB9
(23E2)/$3E/$42 LD A,42H
(23E4)/$D3/$96 OUT (P100D),A
(23E6)/$AF XOR A
(23E7)/$2F CPL
(23E8)/$E6/$1F AND 1FH
(23EA)/$D3/$92 OUT (S102D),A
(23EC)/$DB/$93 (SCHREIBA: IN A,(S102C)
(23EE)/$C0/$57 BIT 2,A
(23F0)/$20/$FA JR NZ,SCHREIBA
(23F2)/$DB/$93 IN A,(S102C)
(23F4)/$32/$2130 LD (PUFAZ),A
(23F7)/$3E/$05 LD A,5
(23F9)/$D3/$93 OUT (S102C),A
(23FB)/$3E/$00 LD A,0
(23FD)/$D3/$93 OUT (S102C),A
(23FF)/$C3/$227F JP STOPF1
(2402)/$21/$210B (SCHREIBB: LD HL,PUFE
(2405)/$3A/$210A LD A,(PUFEZ)
(2408)/$FE/$26 CR 26H
(240A)/$38/$2C JP C,FALSCH
(240C)/$47 LD B,A
(240D)/$7E (SCHREIBC: LD A,(HL)
(240E)/$C0/$243F CALL PRUEF
(2411)/$CA/$241C JP Z,RIGHTIG
(2414)/$3E/$03 LD A,3
(2416)/$32/$2109 LD (FESP),A
(2419)/$C3/$2448 JP FEHLER
(241C)/$E6/$0F (RIGHTIG: AND 0FH
(241E)/$E2/$2423 JP PO,SCHREIBD

```

```

(2421)/$C6/$10      {
(2423)/$77           {SCHREIBD: ADD A,10H
(2424)/$23           { LD (HL),A
(2425)/$10/$E6       { INC HL
(2427)/$3A/$210A     { DJNZ SCHREIBD
(2428)/$47           { LD A,(P0FEZ)
(2428)/$3E/$25       { LD B,A ; freie Stellen bis 37 Zeichen
(242D)/$9B           { LD A,25H ; mit 00H auffuellen
(242E)/$47           { SUB B
(242E)/$47           { LD B,A
(242F)/$3E/$0D       { LD A,00DH
(2431)/$77           {SCHREIBE: LD (HL),A
(2432)/$23           { INC HL
(2433)/$10/$FC       { DJNZ SCHREIBE
(2435)/$C3/$2321     { JP SCHREIBF
(2438)/$3E/$04       {FALSCH: LD A,4 ; unerlaubte Zeichenanzahl
(243A)/$32/$2109     { LD (FESP),A
(243D)/$10/$09       { JR FEHLER
}

; Zeichenpruefung
(243F)/$21/$2559     {PRUEF: LD HL,ZPRUEF ; ZPRUEF ist Zeichentabelle
(2442)/$01/$000C     { LD BC,12 ; fuer erlaubte Zeichen
(2445)/$ED/$B1       { CPIR
(2447)/$C9           { RET
}

; Fehlerroutine
(2448)/$0E/$09       {FEHLER: LD C,9 ; Fehlerausgabe nur bei TEST !!
(244A)/$3A/$21A8     { LD A,(MERK)
(244D)/$FE/$53       { CP 53H
(244F)/$CA/$227F     { JP Z,STOPF1
(2452)/$FE/$4C       { CP 4CH
(2454)/$CA/$227F     { JP Z,STOPF1
(2457)/$3E/$30       { LD A,30H
(2459)/$D3/$93       { OUT (SID2C),A
(245B)/$3A/$2109     { LD A,(FESP)
(245E)/$FE/$01       { CP 1
(2460)/$28/$10       { JR Z,FEHLER1 ; Lesefehler
(2462)/$FE/$02       { CP 2
(2464)/$28/$11       { JR Z,FEHLER2 ; 2. Karte
(2466)/$FE/$03       { CP 3
(2468)/$28/$12       { JR Z,FEHLER3 ; unerlaubte Zeichen
(246A)/$FE/$04       { CP 4
(246C)/$28/$13       { JR Z,FEHLER4 ; unerlaubte Laenge
(246E)/$AF           { XOR A
(246F)/$32/$2109     { LD (FESP),A
(2472)/$11/$2565     {FEHLER1: LD DE,FE1
(2475)/$18/$0D       { JR FEHLERE
(2477)/$11/$257E     {FEHLER2: LD DE,FE2
(247A)/$18/$08       { JR FEHLERE
(247C)/$11/$259D     {FEHLER3: LD DE,FE3
(247F)/$18/$03       { JR FEHLERE
(2481)/$11/$25B4     {FEHLER4: LD DE,FE4
(2484)/$CD/$0005     {FEHLERE: CALL BDOS
(2487)/$C3/$227F     { JP STOPF1
}

; Programmende
(248A)/$C3/$21D4     {ENDE: JP START0
(248D)/$ED/$7B/$2157 {ENDE1: LD SP,(SPA)
(2491)/$C3/$0000     { JP 0
}

; Zeitkonstante fuer SIO
(2494)/$3E/$05       {ZK: LD A,5
(2496)/$D3/$9B       { OUT (CTCK3),A
(2498)/$AF           { XOR A
(2499)/$D3/$9B       { OUT (CTCK3),A
(249B)/$DB/$9B       { IN A,(CTCK3)
(249D)/$3E/$03       { LD A,3
(249F)/$D3/$9B       { OUT (CTCK3),A
(24A1)/$C9           { RET
}

; Startinitialisierung
(24A2)/$CD/$24A6     {INIT: CALL INIT
(24A5)/$C9           { RET
(24A6)/$21/$2657     {INI: LD HL,P10AC ; P10 Opto-Koppler
(24A9)/$01/$0395     { LD BC,0300H+P100KC; (P10A)
(24AC)/$ED/$B3       { OTIR
(24AE)/$DB/$94       { IN A,(P100KD) ; P10A Start
(24B0)/$3E/$0F       { LD A,00FH ; Byteausgabe
(24B2)/$D3/$97       { OUT (P100C),A ; Motorsteuerung
(24B4)/$3E/$03       { LD A,3 ; Interrupt gesperrt
(24B6)/$D3/$97       { OUT (P100D),A
(24B8)/$3E/$47       { LD A,47H
(24BA)/$D3/$96       { OUT (P100D),A
(24BC)/$21/$2651     { LD HL,CTC0 ; CTC-Kanal 0
(24BF)/$01/$0298     { LD BC,0200H+CTC0
(24C2)/$ED/$B3       { OTIR
(24C4)/$3E/$03       { LD A,3
(24C6)/$D3/$99       { OUT (CTCK1),A ; CTC-Kanal 1
}

```

← RET!!!


```

(24C8)/$D3/$4A      {
(24CA)/$D3/$4B      {
(24CC)/$3E/$10      {
(24CE)/$D3/$43      {
(24D0)/$D3/$41      {
(24D2)/$21/>>$2666  {
(24D5)/$01/>>$0E93  {
(24D8)/$ED/$B3      {
(24DA)/$21/>>$265A  {
(24DD)/$01/>>$0C91  {
(24E0)/$ED/$B3      {
(24E2)/$3A/>>$21A8  {
(24E5)/$FE/$53      {
(24E7)/$20/$08      {
(24E9)/$21/>>$2674  {INIB:
(24EC)/$01/>>$0A93  {
(24EF)/$ED/$B3      {
(24F1)/$C9          {INIA:
(24F2)/$21/>>$264B  {INI1:
(24F5)/$01/>>$0298  {
(24F8)/$ED/$B3      {
(24FA)/$21/>>$267E  {INI2:
(24FD)/$01/>>$0493  {
(2500)/$18/$21      {
(2502)/$CD/>>$250D  {INI16:
(2505)/$21/>>$2682  {INI7:
(2508)/$01/>>$0893  {
(250B)/$18/$16      {
(250D)/$21/>>$2653  {INI9:
(2510)/$01/>>$0298  {
(2513)/$ED/$B3      {
(2515)/$21/>>$2655  {INI10:
(2518)/$01/>>$0299  {
(251B)/$ED/$B3      {
(251D)/$21/>>$264D  {INI11:
(2520)/$01/>>$029A  {
(2523)/$ED/$B3      {INI13A:
(2525)/$C9          {
(2526)/$21/>>$264F  {INI22:
(2529)/$01/>>$029B  {
(252C)/$18/$F5      {
(252E)/$3A/>>$21A8  {BILD:
(2531)/$FE/$54      {
(2533)/$28/$01      {
(2535)/$C9          {
(2536)/$0E/$09      {BILD1:
(2538)/$3A/>>$21B1  {
(253B)/$FE/$01      {
(253D)/$28/$09      {
(253F)/$FE/$02      {
(2541)/$28/$0A      {
(2543)/$FE/$03      {
(2545)/$28/$0B      {
(2547)/$C9          {
(2548)/$11/>>$2649  {BILD2:
(254B)/$18/$08      {
(254D)/$11/>>$2604  {BILD3:
(2550)/$18/$03      {
(2552)/$11/>>$2601  {BILD4:
(2555)/$CD/>>$0005  {BILD5:
(2558)/$C9          {
(
(
(
(
(2559)/$30/$31/$32/$33 {ZPRUEF:
(255D)/$34/$35/$36/$37
(2561)/$38/$39      {
(2563)/$20/$0D      {
(
(
(
(
(2565)/$40/$0A/$4C/$20 {FE1:
(2569)/$45/$20/$53/$20
(256D)/$45/$20/$46/$20
(2571)/$45/$20/$48/$20
(2575)/$4C/$20/$45/$20
(2579)/$52/$20/$0D/$0A
(257D)/$24          {
(257E)/$40/$0A/$4B/$61 {FE2:
(2582)/$72/$74/$65/$20
(2586)/$69/$6D/$20/$47
(258A)/$65/$72/$61/$65
(258E)/$74/$20/$67/$65
(2592)/$77/$65/$73/$65
(2596)/$6E/$20/$21/$21
(259A)/$40/$0A/$24  {
(259D)/$40/$0A/$75/$6E {FE3:
(25A1)/$65/$72/$6C/$61
(25A5)/$75/$62/$74/$65
(25A9)/$20/$5A/$65/$69
(25AD)/$63/$68/$65/$6E
(25B1)/$40/$0A/$24

```

```

(25D4)/$0D/$0A/$75/$6E (FE4: DB CR,LF,'unerlaubte Laenge (>37 Zeichen!),'CR,LF,'$')
(25D8)/$65/$72/$6C/$61
(25DC)/$75/$62/$74/$65
(25E0)/$20/$4C/$61/$65
(25E4)/$6E/$67/$65/$20
(25E8)/$28/$3E/$33/$37
(25EC)/$20/$5A/$65/$69
(25F0)/$63/$68/$65/$6E
(25F4)/$21/$29/$0D/$0A
(25F8)/$24
(25FC)/$0D/$0A/$45/$69 (LES: DB CR,LF,'Eingabe L fuer Lesen, S fuer Schreiben')
(2600)/$6E/$67/$61/$62
(2604)/$65/$20/$4C/$20
(2608)/$66/$75/$65/$72
(260C)/$20/$4C/$65/$73
(2610)/$65/$6E/$2C/$20
(2614)/$53/$20/$66/$75
(2618)/$65/$72/$20/$53
(261C)/$63/$68/$72/$65
(2620)/$69/$62/$65/$6E
(2624)/$0D/$0A/$24 (LER: DB CR,LF,'$'
(2628)/$0D/$0A/$0D/$0A (TEXT: DB CR,LF,CR,LF,'> M A G N E T - K E N N K A R T E N <'')
(262C)/$3E/$20/$4D/$20
(2630)/$41/$20/$47/$20
(2634)/$4E/$20/$45/$20
(2638)/$54/$20/$2D/$20
(263C)/$4B/$20/$45/$20
(2640)/$4E/$20/$4E/$20
(2644)/$4B/$20/$41/$20
(2648)/$52/$20/$54/$20
(264C)/$45/$20/$4E/$20
(2650)/$3C
(2654)/$0D/$0A/$20/$20 ( DB CR,LF,' lesen/schreiben',CR,LF,'$')
(2658)/$20/$20/$20/$20
(265C)/$20/$20/$4C/$65
(2660)/$73/$65/$6E/$2F
(2664)/$73/$63/$68/$72
(2668)/$65/$69/$62/$65
(266C)/$6E/$0D/$0A/$24
(2670)/$0C/$24 (LOE: DB CLS,'$'
)
)
E/A-Daustein-Programmierung
)
(264B)/$27/$0A (CTC00: DB 27H,00AH
(264D)/$07/$02 (CTC02: DB 07H,02H
(264F)/$35/$F0 (CTC03: DB 35H,0F0H
(2651)/$D0/$03 (CTC0: DB 0D0H,03H
(2653)/$47/$F0 (CTC10: DB 47H,0F0H
(2655)/$47/$47 (CTC110: DB 47H,47H
(2657)/$CF/$07/$07 (PIOAC: DB 0CFH,07H,07H
(265A)/$04/$10/$01/$00 (SIOAC: DB 4,10H,1,0,3,2,5,0,6,0,7,58H
(265E)/$03/$02/$05/$00
(2662)/$06/$00/$07/$58
(2666)/$04/$10/$01/$00 (SIOBCL: DB 4,10H,1,0,3,2,5,0,6,0,7,58H,2,00CH
(266A)/$03/$02/$05/$00
(266E)/$06/$00/$07/$58
(2672)/$02/$0C
(2674)/$04/$10/$06/$00 (SIOBC: DB 4,10H,6,0,7,58H,1,0,5,8AH
(2678)/$07/$58/$01/$00
(267C)/$05/$8A
(267E)/$01/$00/$05/$03 (SIO1AC: DB 1,0,5,3
(2682)/$03/$11/$01/$00 (SIO1DC: DB 3,11H,1,0,6,0,7,58H
(2686)/$06/$00/$07/$58
)
END

```

19.06.89 / 12:43

-01-

{KKDef.inc Definitionen fuer KKarte; Pr/30.5.89}

```
const
  Version='Kennkarten Schreiben    V 1.0';

  MeldZ = 20;      {Zeilen-Nr fuer Bedienermeldung und Fehleranzeige}
  FehlZ = 22;

  FehlerDatei= -1000;      {Kennzahl Dateifehler}

  Return = ^M;      {Tastencodes}
  Links = ^H;      {Test}
  Rechts = ^D;
  Reset = ^C;
  DEL = ^G;

  KKZeichen= ' @123456789';      {gueltige Zeichen fuer Kennkarten-Schreiben}
type
  Str2 = String[2];
  Str10 = String[10];
  Str30 = String[30];
  Str37 = String[37];
  Str14 = String[14];
  Str64 = String[64];
  Str80 = String[80];

  KKTyp = Str37;
  FDateiTyp = Str80;
  DateiNameTyp= (Str14, Str80);

  MenueTyp = (Menue, Txt, Dbf);
  AktionTyp= (Lesen, Schreiben);
  ParamTyp = record
    Ueberschrift : Str64;
    case ArtMenue: MenueTyp of
      Menue: (Aktion: AktionTyp);
      Txt : (DateiTxt: DateiNameTyp);
      Dbf : (DateiDbf: DateiNameTyp);
    FAnzeige, FKKarte, FBedingung: Str10;
  end(record);

var
  Param: ParamTyp;
```

Anzeige → Beschriftung !

(KKSle.inc Treiber SLE Pr/30.5.89)
 (MC-Programm von Bus 12.5.89)

```

const
  (SLE-Treiber)
    AdrTest = $2100;
    AdrRead = $2103;
    AdrWrite = $2106;
    AdrError = $2109;
    AdrPufW = $210C;
    AdrPufR = $2112;
    (Procedur-Adresse)
    (Fehlerbyte des Treibers)
    (Schreibpuffer)
    (Lesebuffer)

type
  KKTyp = String[37];
  (Fehlermeldungen)
  KKErrorTyp = (OK,
                lesefeh!,
                karte,
                unzu!aessig,
                laenge,
                ungleich);
                (Aktion korrekt)
                (Lesefehler)
                (Karte noch im Geraet)
                (unzu!aessige Zeichen beim Schreiben)
                (unerlaubte Laenge)
                (Kontroll-Lesen nicht mit Schreiben identisch)

var
  PufError: KKErrorTyp (absolute AdrError);
  PufWrite: KKTyp (absolute AdrPufW);
  PufRead: KKTyp (absolute AdrPufR);

procedure KKTest;      external AdrTest;      (Adressen in SLE.inl)
procedure KKRead;      external AdrRead;
procedure KKWrite;     external AdrWrite;

function KKLesen(var KennKarte: KKTyp): KKErrorTyp;
begin
  KKRead;
  if Length(PufRead) < SizeOf(KKTyp) then
    KennKarte := PufRead;
  KKLesen := PufError;
end(KKLesen);

function KKSchreiben(var KennKarte: KKTyp): KKErrorTyp;
begin
  PufWrite := KennKarte;
  KKWrite;
  KKSchreiben := PufError;
  if PufError = OK then begin
    KKRead;
    KKSchreiben := PufError;
    if PufError = OK then begin
      while Length(PufWrite) < Length(PufRead) do
        PufWrite := PufWrite + ' ';
      if PufRead <> PufWrite then
        KKSchreiben := ungleich;
    end(if);
  end(if);
end(KKSchreiben);

function TestMCProg: Boolean;
(coup: true: SLE-Treiber ist geladen)

const
  Jmp = $C3;

begin
  TestMCProg := false;
  if (Addr(KKRead) <> AdrRead) or (Addr(KKWrite) <> AdrWrite) then
    Writeln('Adr Read: ', Addr(KKRead), ' Soll-Adr: ', AdrRead)
  else
    if (Mem(AdrWrite) <> Jmp) or (Mem(AdrRead) <> Jmp) then
      Writeln('kein JMP')
    else
      TestMCProg := true;
end(TestMCProg);

```

(KKBild.inc Bildschirm fuer KKarte
(benoetigt KKDef)

Pr/30.5.89)

procedure LoeZeile(Zeile: Byte);

```
begin
  GotoXY(1,Zeile);
  ClrEol;
end;
```

procedure BildSchirm(Ueberschrift: Str64);
(Aufbau des Anfangs-Bildschirmes)

```
  procedure Linie;
  begin
    Writeln('*****');
  end(Linie);

begin
  ClrScr;
  Writeln(Versfon);
  Writeln;
  Linie;
  GotoXY(40-Length(Ueberschrift) div 2,4);
  Write(Ueberschrift);
  GotoXY(1,4);
  Write('#');
  GotoXY(80,4);
  Write('#');
  Linie;
  Writeln;
end(BildSchirm);
```

procedure Ende;
(Aufbau des End-Bildschirmes)

```
begin
  BildSchirm('Ende Kennkarten - Schreiben');
end;
```

procedure FehlerMeld(Fehler: KKErrortyp);
(Ausgabe Fehlermeldung auf aktuelle Zeile)

```
begin
  case Fehler of
    lesefehl : Write('Lesefehler');
    karte : Write('Karte noch im Geraet');
    unzulessig: Write('nur Ziffern erlaubt');
    laenge : Write('zu viele Zeichen');
    ungleich : Write('Schreibfehler');
  end(case);
  if Fehler <> OK then
    Write(BEL);
  end(FehlerMeld);
```

function Meldung(Meld: Str30; Fehler: KKErrortyp): Boolean;
(Ausgabe einer Bedienerfuehrung auf Zeile MeldZ und Fehlermeldung auf
Zeile FehlZ; Warten auf Bestaetigung)
(oup: true: positive; false: negative Bestaetigung)

```
var
  Taste: Char;
begin
  LoeZeile(MeldZ);
  GotoXY(1,MeldZ);
  Write(Meld);
  GotoXY(1,FehlZ);
  FehlerMeld(Fehler);
  GotoXY(50,MeldZ);
  Write('Ausgefuehrt: (J)a/(N)ein: J',Links);
  repeat
    Read(kbd,Taste);
    if Taste=Return then
      Taste:='J';
    Taste:=UpCase(Taste);
  until (Taste='J') or (Taste='N');
  LoeZeile(FehlZ); LoeZeile(MeldZ);
  GotoXY(1,MeldZ);
  Meldung:=Taste='J';
end(Meldung);
```

Ausführung

2A

10.06.1989

{dBase2.inc Zugriff zu dBASE2-Dateien mit Satzlaengen<=2040}

```

const
  dBPuflen = 2047;
  BlockLen = 128;
  dBKopfLen = 520;
  MaxFeldLen = 80;
  dBDelete = '*';
  dBEnde = '^';

type
  (
    DateNameTyp = String[14];
    FDateTyp = string[MaxFeldLen];

    FeldNameTyp = string[10];
    dBFeldTyp = record
      FeNa: ARRAY[1..10] of char;
      DBy: Byte;
      FLy: Char;
      FLen: Byte;
      FPos: Integer;
      DBy: Byte;
    end;
    dBFileType = record
      Name: DateNameTyp;
      dBFile: File;
      dBPuf: array[0..dBPuflen] of Char;
      BlockNr: Integer;
      SatzPos: Integer;
      SatzNr: Integer;
      Puflen: Integer;
      IstWrite: Boolean;
      Eof: Boolean;
      dBK: record
        KByte: Byte;
        SatzAnz: Integer;
        dBDatum: array[1..3] of Char;
        SatzLen: Integer;
        dBFeld: array[1..32] of dBFeldTyp;
      end;
    end;

  var
    dB2: dBFileType;
    dBFile: File;
    dBPuf: array[0..dBPuflen] of Char;

  procedure dBNextSatz;
  {Naechste Satz}

  var
    AltPos, b: Integer;

  begin
    with dB2 do begin
      AltPos := SatzPos;
      if SatzNr > 0 then
        SatzPos := SatzPos + dBK.SatzLen;
      SatzNr := Succ(SatzNr);
      if SatzPos > dBPuflen then begin
        if IstWrite then begin
          Seek(dBFile, BlockNr);
          BlockWrite(dBFile, dBPuf, Puflen div BlockLen);
          IstWrite := false;
        end;
        BlockNr := BlockNr + (SatzPos div BlockLen);
        Seek(dBFile, BlockNr);
        BlockRead(dBFile, dBPuf, Succ(dBPuflen) div BlockLen, b);
        Puflen := b * BlockLen;
        SatzPos := SatzPos mod BlockLen;
      end;
    end;
  end;

  function dBSkip: Boolean;
  {naechste Satz mit uebergehen geloeschter Satze; true: noch Daten vorhanden}

  begin
    with dB2 do begin
      repeat
        dBNextSatz;
      until (dBPuf[SatzPos] < dBDelete) or (dBPuf[SatzPos] = dBEnde) or (Puflen = 0);
      Eof := (dBPuf[SatzPos] = dBDelete) or (Puflen = 0);
      dBSkip := not Eof;
    end;
  end;

```

```
function dBOpen(dBDateName: DateNameTyp): Boolean;
  {oeffnen einer dBase2-Datei; true: geoeffnet; false: keine Datei, nicht dBase2}
```

```
var
  i,p,b: Integer;
```

```
begin
  dBOpen:=false;
  i:=1;
  if Pos('.', dBDateName)=0 then dBDateName:=dBDateName+'.dbf';
  Assign(dBFile, dBDateName);
  {#1-} Reset(dBFile); {#1-}
  if IOResult=0 then begin {wenn File existiert}
    with dB2 do begin
      BlockRead(dBFile, dBPuf, Succ(dBPufLen) div BlockLen, b);
      if b)Succ(dBKopfLen div BlockLen) then begin {und volle Laenge}
        Move(dBPuf, dBK, SizeOf(dBK));
        if (dBK.KByte=2) and (dBK.SatzLen<=dBPufLen-BlockLen) then begin {und Kennbyte
```

```
OK)
        p:=Pred(dBK.dBFeld[i].FPos);
        repeat
          dBK.dBFeld[i].FPos:=Abs(dBK.dBFeld[i].FPos-p); {Pos. Feld im Satz}
          i:=Succ(i);
        until (i>32) or (dBK.dBFeld[i].BBy>0); {und Blindbyte alle 0=}
        if i>32 then begin
          IstWrite:=false;
          Eof:=false;
          SatzPos:=Succ(dBKopfLen);
          BlockNr:=0;
          dB2.PufLen:=b*BlockLen;
          SatzNr:=0;
          dBOpen:=dBSkip; {evt. geloeschte Saetze uebergangen, oder Puffer richtig fuellen}
        end{if};
      end{if};
      if i<=32 then Close(dBFile);
    end{with};
  end{if};
end{dBOpen};
```

```
procedure dBClose;
  {Schliessen der dBase-Datei}
```

```
begin
  if dB2.IstWrite then begin {erst zurueckschreiben}
    Seek(dBFile, dB2.BlockNr);
    BlockWrite(dBFile, dBPuf, dB2.PufLen div BlockLen);
    dB2.IstWrite:=false;
  end{if};
end;
```

```
function dBEOF: Boolean;
  {Eof des dBasefiles}
```

```
begin
  dBEOF:=dB2.Eof;
end;
```

```
procedure dBFieldParam(Fieldname: FieldNameTyp; var FieldPos, FieldLen: Integer);
  {ermittelt die Parameter des Feldes Fieldname; FieldPos: Position im Puffer;
  wenn Fieldname nicht vorhanden, dann FieldLen=0}
```

```
var
  i: Integer;
  FN: array[1..10] of Char;
```

```
begin
  i:=1;
  while (i<=Length(Fieldname)) and (i<=10) do begin
    FN[i]:=UpCase(Fieldname[i]);
    i:=Succ(i);
  end;
  while i<=10 do begin
    FN[i]:=^0; {fuellen mit BlindByte}
    i:=Succ(i);
  end{while};
  i:=0;
  with dB2.dBK do begin
    repeat
      i:=Succ(i);
    until (i=32) or (dBField[i].FeNa=FN);
    if dBField[i].FeNa=FN then begin
      FieldPos:=dBField[i].FPos+dB2.SatzPos;
      FieldLen:=dBField[i].FLen;
    end{then}
  end;
  else
    FieldLen:=0;
  end{with};
end{dBFieldParam};
```

```
function dBRead(FeldName:FeldNameTyp): FDatenTyp;
  (Lesen des Inhaltes eines Feldes)
```

```
var
  FLen, FPos: Integer;
  Daten      : FDatenTyp;
begin
  with dB2 do begin
    if not Eof then begin
      dBFieldParam(FeldName, FPos, FLen);
      if FLen=0 then
        Daten:=
      else begin
        if FLen>MaxFeldLen then FLen:=MaxFeldLen;
        Daten[0]:=Char(FLen);
        Move(dBPuf[FPos], Daten(1, FLen);
      end(if);
    end(then)
  else
    Daten:='';
  end(with);
  dBRead:=Daten;
end(dBRead);
```

```
procedure dBWrite(FeldName:FeldNameTyp; FeldDaten: FDatenTyp);
  (Schreiben des Feldes in den aktuellen Satz)
```

```
var
  FLen, FPos, l: Integer;
begin
  with dB2 do begin
    if not Eof then begin
      dBFieldParam(FeldName, FPos, FLen);
      if FLen=0 then
        FeldDaten:=
      else begin
        l:=Length(FeldDaten);
        if l>FLen then l:=FLen;
        Move(FeldDaten(1), dBPuf[FPos], l);
        while l<FLen do begin
          dBPuf[FPos+l]:=
          l:=Succ(l);
        end(while);
        IstWrite:=true;
      end(if);
    end(then)
  else
    FeldDaten:='';
  end(with);
end(dBWrite);
```

```
(Test:
var f:fdatentyp;
begin
if dBOpen('d:\dbase2\neulos') then begin
write(db2, satznr, ' ');
readfeld('ag', f); write(f, ' ');
readfeld('ma', f); write(f, ' ');
readfeld('losnr', f); write(f, ' ');
readfeld('datum', f); write(f, ' ');
readfeld('zeit', f); write(f, ' ');
writeln;
dbclose;
end;
end;
)
```


{TxtDat.inc Lesen einer Text-Datendatei Pr/6.6.89}
 {Textdatei besteht aus Zeilen, deren Datenfelder mit Delimiter aufgebaut sind.
 Beispiel: ... ; ... ;}

```
const
  DelStr = ' ';
  DelFeld = ' ';

type
  FDatenTyp = String[80];
  FNameTyp = String[14];
var
  TxtFile : Text;
  TxtPuf : String[255];

function TxtSkip: Boolean;
  {naechste Datensatz; true: Daten vorhanden}

begin
  if Eof(TxtFile) then
    TxtPuf := '';
  else
    Readln(TxtFile, TxtPuf);
    TxtSkip := Length(TxtPuf);
  end(TxtSkip);

function TxtOpen(DateiName: FNameTyp): Boolean;
  {oeffnen der Text-Datendatei; true: geoeffnet; false: keine Datei, Diskfehler,
  falsche Textformat}

begin
  TxtOpen := false;
  Assign(TxtFile, DateiName);
  {$I-} Reset(TxtFile); {$I+}
  if IOResult = 0 then begin
    TxtOpen := TxtSkip;
  end(if);
end(TxtOpen);

function TxtEof: Boolean;
  {Test Dateiende}

begin
  TxtEof := Eof(TxtFile);
end(TxtEof);

procedure TxtFeldParam(FeldNr: Byte; var FeldPos, FeldLen: Integer);
  {ermittelt die Position und die Laenge des Feldes mit FeldNr im Puffer;
  ist dieses Feld nicht vorhanden; dann FeldLen:=0}

var
  l: Byte;

begin
  l := Length(TxtPuf);
  if (FeldNr > 0) and (l > 0) then begin
    FeldPos := 1;
    repeat
      if FeldPos > l then begin
        FeldPos := FeldPos + FeldLen;
        if TxtPuf[FeldPos] = DelStr then FeldPos := Succ(FeldPos);
        if TxtPuf[FeldPos] = DelFeld then FeldPos := Succ(FeldPos);
      end(if);
      if TxtPuf[FeldPos] = DelStr then begin
        FeldPos := Succ(FeldPos);
        FeldLen := Pos(DelStr, Copy(TxtPuf, FeldPos, l - FeldPos + 1)) - 1;
      end(then)
      else begin
        FeldLen := Pos(DelFeld, Copy(TxtPuf, FeldPos, l - FeldPos + 1)) - 1;
        if FeldLen < 0 then FeldLen := l - FeldPos + 1;
      end(if);
      FeldNr := Pred(FeldNr);
    until (FeldNr = 0) or (FeldPos + FeldLen > l) or (FeldLen < 0);
    if (FeldNr < 0) or (FeldLen < 0) then FeldLen := 0;
  end(then)
  else
    FeldLen := 0;
  end(TxtFeldParam);

function TxtRead(FeldNr: Byte): FDatenTyp;
  {Lesen des Feldes; Feld nicht vorhanden: Leerstring}

var
  FeldPos, FeldLen: Integer;
```

```
begin
  TxtFeldParam(FeldNr,FeldPos,FeldLen);
  if FeldLen>0 then
    TxtRead:=Copy(TxtPuf,FeldPos,FeldLen)
  else
    TxtRead:='';
  end(TxtRead);
```

{KKMenu.inc Menu-Fuehrung fur Kennkarten Pr/7.6.89)
 (benoetigt KKDef, KKBild)
 (Aufbau wie Dateizugriffe dBase2 und TxtDat)

```
var
  MenEnd: Boolean;
  MenPuf: KKTyp;

function MenEingabe: Boolean;
  (Eingabe fuer das Schreiben; false: Abbruch mit Reset).

var
  p: Byte;
  Taste: Char;

begin
  MenPuf:= '';
  for p:=1 to SizeOf(MenPuf)-1 do MenPuf:=MenPuf+' '; {Puffer mit Leerzeichen fuehlen}
  GotoXY(1,9);
  Writeln('Eingabe der Daten fuer die Kennkarte');
  Writeln('max. ', Length(MenPuf), ' Ziffern; RETURN: bestaetigt; RESET: kein Schreiben');
  p:=1;
  repeat
    GotoXY(20,12);
    Write('!', MenPuf, ' Zeichen: ', p:3);
    GotoXY(20+p,12); {Cursor setzen; p: Position}
    repeat
      Read(kbd, Taste);
    until Pos(Taste, KKZeichen+Links+Rechts+Return+Reset) > 0;
    case Taste of
      Links: if p>1 then p:=Pred(p);
      Rechts: if p<Length(MenPuf) then p:=Succ(p);
      Return: ;
      Reset: ;
    else begin
      MenPuf[p]:=Taste;
      if p<Length(MenPuf) then p:=Succ(p); {Zeichen eintragen}
    end;
  end;
  until Pos(Taste, Return+Reset) > 0;
  for p:=9 to 12 do LoeZeile(p);
  MenEingabe:=Taste=Return;
end(MenEingabe);
```

function MenSkip: Boolean;
 (Entscheidung Lesen/Schreiben/Beenden; true: nicht beenden)

```
var
  Taste: Char;
  i: Byte;
  Abbruch: Boolean;

begin
  LoeZeile(15);
  repeat
    GotoXY(1,8);
    Writeln('Waehlen: L lesen');
    Writeln('S schreiben');
    Writeln('E ende');
    repeat
      Read(kbd, Taste);
      Taste:=UpCase(Taste);
    until Pos(Taste, 'LSE') > 0;
    for i:=8 to 11 do LoeZeile(i);
    Abbruch:=true;
    case Taste of
      L: Param.Aktion:=Lesen;
      S: begin
          Param.Aktion:=Schreiben;
          Abbruch:=MenEingabe;
        end;
    end;
  end;
  until Abbruch;
  MenEnd:=Taste='E';
  MenSkip:=MenEnd;
end(MenSkip);
```

function MenRead: FDatenTyp;
 (holen der eingegebenen Daten)

```
begin
  MenRead:=Copy(MenPuf, 1, Length(MenPuf));
end(MenRead);
```

```
function MenEof: Boolean;  
(true; beenden)
```

```
begin  
  MenEof:=MenEnd;  
end(MenEof);
```

```
function MenOpen: Boolean;
```

```
begin  
  MenEnd:=false;  
  MenOpen:=MenSkip;  
end(MenOpen);
```

```
{KKPar.inc, Parameteranalyse fuer KKarte Pr/30.5.89}
{benoetigt KKDef}
```

```
function ParamAnalyse(var Param: ParamTyp): Boolean;
{Analyse der uebergebenen Parameter entsprechend der Syntax; true: Syntax ok}
```

```
var
  Pc, i: Byte;
  Ps: Str2;
begin
  ParamAnalyse:=false;
  Pc:=ParamCount;
  if (Pc>1) and (Length(ParamStr(1))=2) and (Copy(ParamStr(1),1,1)='/') then begin
    Ps:=ParamStr(1);
    case UpCase(Ps[2]) of
      'M': if Pc=1 then begin
            Param.ArtMenue:=Menue;
            Param.Ueberschrift:='Lesen oder Schreiben';
            ParamAnalyse:=true;
          end;
      'T': if Pc=3 then begin
            Param.ArtMenue:=Txt;
            Param.Ueberschrift:=ParamStr(2);
            Param.DateiTxt:=ParamStr(3);
            ParamAnalyse:=true;
          end;
      'D': if (Pc=5) or (Pc=6) then begin
            Param.ArtMenue:=dB;
            Param.Ueberschrift:=ParamStr(2);
            Param.DateiDbf:=ParamStr(3);
            Param.FAnzeige:=ParamStr(4);
            Param.FKKarte:=ParamStr(5);
            if Pc=6 then
              Param.FBedingung:=ParamStr(6)
            else
              Param.FBedingung:='';
            ParamAnalyse:=true;
          end;
    end;
  end;
  if Pc=2 then
    for i:=1 to Length(Param.Ueberschrift) do
      if Param.Ueberschrift[i]=' ' then
        Param.Ueberschrift[i]:='';
    end;
  else
    ParamAnalyse:=false;
  end;
end; {ParamAnalyse};
```

{KKArb.inc Bearbeitungsprozeduren fuer KKarte Pr/30.5.89}
 {benoetigt KKDef, KKBild, KKMen, dBase2, TxtDaten}

procedure KarteSchreiben(KKarte: KKTyp);
 {Schreiben einer Karte}

var
 Fehler: KKErrortyp;

begin
 GotoXY(1,17);
 Writeln('Schreiben der Kennkarte');
 Fehler:=OK;
 repeat
 if Meldung('Karte in SLE einschieben',Fehler) then begin
 Write('Schreiben laeuft');
 Fehler:=KKSchreiben(KKarte);
 if Fehler<>OK then begin
 if not Meldung('Nochmal Schreiben?',Fehler) then
 Fehler:=OK;
 end{if};
 end{if};
 until Fehler=OK;
 LoeZeile(17);
 end{Schreiben};

procedure KarteLesen(var KKarte: KKTyp);
 {Lesen einer Karte}

var
 Fehler: KKErrortyp;

begin
 GotoXY(1,17);
 Writeln('Lesen der Kennkarte');
 Fehler:=OK;
 repeat
 if Meldung('Karte in SLE einschieben',Fehler) then begin
 Write('Lesen laeuft');
 Fehler:=KKLesen(KKarte);
 if Fehler<>OK then begin
 if not Meldung('Nochmal Lesen?',Fehler) then
 Fehler:=OK;
 end{if};
 end{if};
 until Fehler=OK;
 LoeZeile(17);
 end{Lesen};

function Naechste: Boolean;
 {Naechste Daten anfordern}

begin
 with Param do begin
 case ArtMenue of
 Menue: Naechste:=MenSkip;
 Txt : Naechste:=TxtSkip;
 Dbf : begin
 if Length(FBedingung)>0 then begin {wenn Bedingung, dann}
 repeat {bis erste Zeichen im Feld}
 until not dBSkip or (Copy(dBRead(FBedingung),1,1)=' ');
 Naechste:=not dBEOF;
 end{then}
 else
 Naechste:=dBSkip;
 end{select};
 end{case};
 end{with};
 end{Naechste};

function Beginnen: Boolean;
 {Beginnen, bzw Dateien eroeffnen}

begin
 with Param do begin
 case ArtMenue of
 Menue: Beginnen:=MenSkip;
 Txt : Beginnen:=TxtOpen(DateiTxt);
 Dbf : begin
 Beginnen:=dBOpen(DateiDbf);
 if not dBEOF and (Length(FBedingung)>0) and (Copy(dBRead(FBedingung),1,1)=' ') then
 Beginnen:=Naechste;
 end{select};
 end{case};
 end{with};
 end{Beginnen};

```
procedure Schliessen;
{Schliessen der Dateien}
```

```
begin
  with Param do begin
    case ArtMenue of
      Txt : TxtClose;
      Dbf : dBClose;
    end(case);
  end(with);
end(Schliessen);
```

```
function Zaehlen: Integer;
{Zaehlen der Daten in den Dateien; wenn <0, dann Dateiprobleme}
```

```
var
  z: Integer;
```

```
begin
  z:=0;
  if Param.ArtMenue<>Menue then begin
    GotoXY(1,6);
    Write('Zaehlen der Karten-Daten in der Datei');
    if Beginnen then begin
      repeat
        z:=Succ(z);
      until not Naechste;
      Schliessen;
      if Beginnen then;
      end(then)
    else
      z:=FehlerDatei;
    end(then)
  else
    z:=-1;
    Zaehlen:=z;
    LoeZeile(6);
  end(Zaehlen);
```

```
procedure DatenHolen(var KKDaten: KKTYP; var Anzeiger: Str80);
{Daten holen}
```

```
begin
  with Param do begin
    case ArtMenue of
      Menue: begin
        Anzeiger:= '';
        if Aktion=Schreiben then
          KKDaten:=MenRead
        else
          KKDaten:= '';
        end(select);
      Txt : begin
        Anzeiger:=TxtRead(1);
        KKDaten:=TxtRead(2);
        end(select);
      Dbf : begin
        Anzeiger:=dBRead(FAnzeiger);
        KKDaten:=dBRead(FKKarte);
        if Length(FBedingung)>0 then (wenn mit Bedingung, dann)
          dBWrite(FBedingung, ''); (Feld leeren)
        end(select);
      end(case);
    end(with);
  end(DatenHolen);
```

```
procedure RearbKarte(KKDaten: KKTYP; Anzeiger: Str80);
{Schreiben oder Lesen ausfuehren}
```

```
begin
  if (Param.ArtMenue=Menue) and (Param.Aktion=Lesen) then begin
    KarteLesen(KKDaten);
    if Length(KKDaten)>0 then begin
      GotoXY(1,17);
      Write('Karteninhalt: ',KKDaten);
    end(if);
  end(then)
  else begin
    GotoXY(1,12);
    WriteLn('Kartenbeschriftung: (Bitte erst Karte beschriften, dann RETURN)');
    WriteLn(Anzeiger);
    WriteLn;
    WriteLn('Karteninhalt: ',KKDaten,'');
    KarteSchreiben(KKDaten);
    LoeZeile(12); LoeZeile(13); LoeZeile(15);
  end(if);
end(Karte);
```

```
procedure KennKarte;  
{Bearbeitung von KKarte; globale Param muss gesetzt sein!}
```

```
var  
  Anz      : Integer;  
  KKDaten: KKTyp;  
  Anzeige: Str80;  
  
begin  
  Bildschirm(Param.Ueberschrift);  
  Anz:=Zaehlen;  
  if Anz=FehlerDatei then begin  
    Bildschirm('Abbruch');  
    GotoXY(1,15);  
    Write('Diskettenfehler, keine Datei oder keine Daten !',BEL);  
    end{then}  
  else begin  
    if Beginnen then begin  
      repeat  
        if Anz>0 then begin  
          GotoXY(1,6);  
          Write('noch zu schreiben:',Anz:4);  
          end{if};  
          Datenholen(KKDaten,Anzeige);  
          BearbKarte(KKDaten,Anzeige);  
          Anz:=Pred(Anz);  
        until (Anz=0) or not Naechste;  
        Schliessen;  
        Bildschirm('Beenden');  
        end{then}  
      else begin  
        Bildschirm('Abbruch');  
        GotoXY(1,15);  
        Write('Diskettenfehler, keine Datei oder keine Daten !',BEL);  
        end{if};  
      end{if};  
    end{KennKarte};
```


(Schreiben und Lesen von Kennkarten mit SLE auf Kopfstation K 8915/ Pr/30.5.89)

{zu Syntax:

- Ueberschrift: Leerzeichen muessen durch ' ' ersetzt werden
- Textdatei : je Zeile: 'Anzeigeinformation', 'Kennkarteninformation'
- dBasedatei : muss dBase 2 sein
die Felder FeldAnzeige und FeldKennkarte muessen vorhanden
sein und vom Typ Zeichen.
wenn das FeldBedingung Zeichen enthaelt, dann wird dieser Satz
benutzt

Weg

(Testversion)

```

program KKarte;
uses crt,turbo3;
(*
procedure SLE;          <Treiber am Anfang>
begin
  inline (
    ($i $1.inl)          <MC-Programm ab 20FFh !>
  );
end(Inlin);
*)
($i KKDef.inc)
($i YKKSle.inc)
($i KKBild.inc)
($i dBase2.inc)
($i TxtDat.inc)
($i KKMenue.inc)
($i KKPar.inc)
($i KKArb.inc)

```

```

begin
  if not ParamAnalyse(Param) then begin
    WriteLn('KKARTE. ---/m-----');
    WriteLn('a) Menue-gesteuertes Lesen und Schreiben. ');
    WriteLn('---/t-Ueberschrift---Textdateiname---');
    WriteLn('b) Schreiben aus der Textdatei ');
    WriteLn('---/d-Ueberschrift---dBaseDateiname---FeldAnzeige->');
    WriteLn('--->FeldKennkarte---');
    WriteLn('---!-FeldBedingung-i');
    WriteLn('c) Schreiben aus dBase-Datei ');
    end{then}
  else
    if TestMCProg then
      KennKarte;
end.

```

*zu kKard.inc
als KK*